



1.7 — Keywords and naming identifiers

👤 ALEX¹ 🕒 MARCH 19, 2024

Keywords

C++ reserves a set of 92 words (as of C++23) for its own use. These words are called **keywords** (or reserved words), and each of these keywords has a special meaning within the C++ language.

Here is a list of all the C++ keywords (through C++23):

alignas

alignof

and

and_eq

asm

auto

bitand

bitor

bool

break

case

catch

char

char8_t (since C++20)

char16_t

char32_t

class

compl

concept (since C++20)

const

constexpr (since C++20)

constexpr

constexpr (since C++20)

constexpr

continue

co_await (since C++20)

co_return (since C++20)

co_yield (since C++20)

decltype

default

delete

do

double

dynamic_cast

else

enum

explicit

export

extern

false

float

for

friend

goto

if

inline

int

long

mutable

namespace

new

noexcept

not

not_eq

nullptr

operator

or

or_eq

private

protected

public

register

reinterpret_cast

requires (since C++20)

return

short

signed

sizeof

static

static_assert

static_cast

struct

switch

template

this

thread_local

throw

true

try

typedef

typeid

typename

union

unsigned

using

virtual

void

volatile

wchar_t

while

xor

xor_eq

The keywords marked (C++20) were added in C++20. If your compiler is not C++20 compliant (or does have C++20 functionality, but it's turned off by default), these keywords may not be functional.

C++ also defines special identifiers: *override*, *final*, *import*, and *module*. These have a specific meaning when used in certain contexts but are not reserved otherwise.

You have already run across some of these keywords, including *int* and *return*. Along with a set of operators, these keywords and special identifiers define the entire language of C++ (preprocessor commands excluded). Because keywords and special identifiers have special meaning, your IDEs will likely change the text color of these words to make them stand out from other identifiers.

By the time you are done with this tutorial series, you will understand what almost all of these words do!

(.)Identifier naming rules [\(#rules\)](#)²

As a reminder, the name of a variable (or function, type, or other kind of item) is called an identifier. C++ gives you a lot of flexibility to name identifiers as you wish. However, there are a few rules that must be followed when naming identifiers:

- The identifier can not be a keyword. Keywords are reserved.
- The identifier can only be composed of letters (lower or upper case), numbers, and the underscore character. That means the name can not contain symbols (except the underscore) nor whitespace (spaces or tabs).
- The identifier must begin with a letter (lower or upper case) or an underscore. It can not start with a number.
- C++ is case sensitive, and thus distinguishes between lower and upper case letters. `nvaLue` is different than `nVaLue` is different than `NVALUE`.

Identifier naming best practices

Now that you know how you *can* name a variable, let's talk about how you *should* name a variable (or function).

First, it is a convention in C++ that variable names should begin with a lowercase letter. If the variable name is a single word or acronym, the whole thing should be written in lowercase letters.

```
1 | int value; // conventional
2 |
3 | int Value; // unconventional (should start with lower case letter)
4 | int VALUE; // unconventional (should start with lower case letter and be in
5 | all lower case)
   | int VaLuE; // unconventional (see your psychiatrist) ;)
```

Most often, function names are also started with a lowercase letter (though there's some disagreement on this point). We'll follow this convention, since function *main* (which all programs must have) starts with a lowercase letter, as do all of the functions in the C++ standard library.

Identifier names that start with a capital letter are typically used for user-defined types (such as structs, classes, and enumerations, all of which we will cover later).

If the variable or function name is multi-word, there are two common conventions: words separated by underscores (sometimes called `snake_case`), or intercased (sometimes called `camelCase`, since the capital letters stick up like the humps on a camel).

```
1  int my_variable_name;    // conventional (separated by
2  underscores/snake_case)
3  int my_function_name(); // conventional (separated by
4  underscores/snake_case)
5
6  int myVariableName;      // conventional (intercapped/camelCase)
7  int myFunctionName();   // conventional (intercapped/camelCase)
8
9  int my variable name;    // invalid (whitespace not allowed)
10 int my function name(); // invalid (whitespace not allowed)
11
12 int MyVariableName;      // unconventional (should start with lower case
13 letter)
14 int MyFunctionName();   // unconventional (should start with lower case
15 letter)
```

In this tutorial, we will typically use the intercased approach because it's easier to read (it's easy to mistake an underscore for a space in dense blocks of code). But it's common to see either -- the C++ standard library uses the underscore method for both variables and functions. Sometimes you'll see a mix of the two: underscores used for variables and intercaps used for functions.

It's worth noting that if you're working in someone else's code, it's generally considered better to match the style of the code you are working in than to rigidly follow the naming conventions laid out above.

Best practice

When working in an existing program, use the conventions of that program (even if they don't conform to modern best practices). Use modern best practices when you're writing new programs.

Second, you should avoid naming your identifiers starting with an underscore, as these names are typically reserved for OS, library, and/or compiler use.

Third, your identifiers should make clear what the value they are holding means (particularly if the units aren't obvious). Identifiers should be named in a way that would help someone who has no idea what your code does be able to figure it out as quickly as possible. In 3 months, when you look at your program again, you'll have forgotten how it works, and you'll thank yourself for picking variable names that make sense.

However, giving a trivial variable an overly complex name impedes overall understanding of what the program is doing almost as much as giving a widely used identifier an inadequate name. Therefore, a good rule of thumb is to make the length of an identifier proportional to how widely it is used. An identifier with a trivial use can have a short name (e.g. such as `i`). An

identifier that is used more broadly (e.g. a function that is called from many different places in a program) should have a longer and more descriptive name (e.g. instead of *open*, try *openFileOnDisk*).

<code>int ccount</code>	Bad	What does the c before “count” stand for?
<code>int customerCount</code>	Good	Clear what we’re counting
<code>int i</code>	Either	Okay if use is trivial, bad otherwise
<code>int index</code>	Either	Okay if obvious what we’re indexing
<code>int totalScore</code>	Either	Okay if there’s only one thing being scored, otherwise too ambiguous
<code>int _count</code>	Bad	Do not start names with underscore
<code>int count</code>	Either	Okay if obvious what we’re counting
<code>int data</code>	Bad	What kind of data?
<code>int time</code>	Bad	Is this in seconds, minutes, or hours?
<code>int minutesElapsed</code>	Good	Descriptive
<code>int value1, value2</code>	Either	Can be hard to differentiate between the two
<code>int numApples</code>	Good	Descriptive
<code>int monstersKilled</code>	Good	Descriptive
<code>int x, y</code>	Either	Okay if use is trivial, bad otherwise

In any case, avoid abbreviations (unless they are common/unambiguous). Although they reduce the time you need to write your code, they make your code harder to read. Code is read more often than it is written, the time you saved while writing the code is time that every reader, including the future you, wastes when reading it. If you’re looking to write code faster, use your editor’s auto-complete feature.

For variable declarations, it is useful to use a comment to describe what a variable is going to be used for, or to explain anything else that might not be obvious. For example, say we’ve declared a variable named `numberOfChars` that is supposed to store the number of characters in a piece of text. Does the text “Hello World!” have 10, 11, or 12 characters? It depends on whether we’re including whitespace or punctuation. Rather than naming the variable `numberOfCharsIncludingWhitespaceAndPunctuation`, which is rather lengthy, a well placed comment on or above the declaration line should help the user figure it out:

```
1 // a count of the number of chars in a piece of text, including whitespace
2 and punctuation
   int numberOfChars;
```

Quiz time

Question #1

Based on how you *should* name a variable, indicate whether each variable name is conventional (follows best practices), unconventional (compiler will accept but does not follow best practices), or invalid (will not compile), and why.

```
int sum {};
```

(Assume it's obvious what we're summing)

[Hide Solution](#) (javascript:void(0))³

Conventional.

```
int _apples {};
```

[Hide Solution](#) (javascript:void(0))³

Unconventional -- variable names should not start with an underscore.

```
int VALUE {};
```

[Hide Solution](#) (javascript:void(0))³

Unconventional -- single word names should be in all lower case.

```
int my variable name {};
```

[Hide Solution](#) (javascript:void(0))³

Invalid -- variable names can not contain spaces.

```
int TotalCustomers {};
```

[Hide Solution](#) (javascript:void(0))³

Unconventional -- variable names should start with a lower case letter.

```
int void {};
```

[Hide Solution](#) (javascript:void(0))³

Invalid -- void is a keyword.

```
int numFruit {};
```

[Hide Solution](#) (javascript:void(0))³

Conventional.

```
int 3some {};
```

[Hide Solution](#) (javascript:void(0))³

Invalid -- variable names can not start with a number.

```
int meters_of_pipe {};
```

[Hide Solution](#) (javascript:void(0))³

Conventional.



[Next lesson](#)

1.8 [Whitespace and basic formatting](#)

4



[Back to table of contents](#)

5



[Previous lesson](#)

1.6 [Uninitialized variables and undefined behavior](#)


6

7



B **U** **URL** **INLINE CODE** **C++ CODE BLOCK** **HELP!**

Leave a comment...

 Name*

@ Email* | ?

Notify me about replies:



POST COMMENT

🔍 Find a mistake? Leave a comment above!?

👤 Avatars from <https://gravatar.com/>¹⁰ are connected to your provided email address.

378 COMMENTS

Newest ▼



Kamil G.

🕒 March 14, 2024 1:14 pm

I personally like/prefer `camelCase` in case of pure letter names and `snake_case` in names with numbers like `case_1`



2



Reply

**Neo**

🕒 March 6, 2024 1:15 am

```

1 | int value; // conventional
2 | int Value; // unconventional (should start with lower case letter)
3 | int VALUE; // unconventional (should start with lower case letter and be in
4 | all lower case)
   | int VaLuE; // unconventional**bold text here**l (see your psychiatrist) ;)

```

see your psychiatrist . . .**No one knows how laughing , I'm Dying Laughing !**

6



Reply

**MonishRules**🗨 Reply to [Neo](#)¹¹ 🕒 March 22, 2024 11:52 pm

i cant afford a psychiatrist so i thought i could write like that



0



Reply

**IceFloe**

🕒 February 18, 2024 4:12 pm

Thanks for the lesson. you correctly noted that variables should be named so that people understand what this or that code is doing. I would also like to add special marks to variables if it has a special meaning, for example, for variables with constant values : double kPi {3.14};



0



Reply

**Kojo Bailey**🗨 Reply to [IceFloe](#)¹² 🕒 April 2, 2024 10:45 pmWhy not just const double pi {3.14};

0



Reply

**trim1**

🕒 February 10, 2024 5:41 am

int VaLuE; // unconventional (see your psychiatrist) ;)

int 3some {};



I fucking love this site



25



Reply

**jbk1703**Reply to [trim1](#)¹³ ⌚ March 12, 2024 6:05 am

LMFAOOO, THEY GOT SOME GOOD SENSE OF HUMOR



1



Reply

**AssAttack**Reply to [trim1](#)¹³ ⌚ March 1, 2024 1:03 am

Agreed.



0



Reply

**cake**Reply to [trim1](#)¹³ ⌚ February 27, 2024 8:49 am

yep, this caught me offguard.



1



Reply

**ElIKyGr**

⌚ January 30, 2024 7:15 am

I think there's a small error regarding underscores

At "Identifier naming rules" states: The identifier must begin with a letter (lower or upper case) or an underscore.

Later in the second point at "Identifier naming best practices":

... you should avoid naming your identifiers starting with an underscore, as these names are typically reserved for OS, library, and/or compiler use.

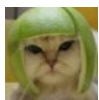
I can tell is for better practices but then, at the earliest point, it should be state it as a side note then further explanation (as in the second point) for the sake of clarity.



1



Reply

**Alex**

Author

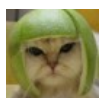
Reply to [ElIKyGr](#)¹⁴ ⌚ January 31, 2024 10:51 pm

The former rule is a must (required by the language), the latter is a should (as doing so is conventional).

 1  Reply**Ajit Mali**

🕒 January 25, 2024 1:30 am

I think the identifiers that start with Capital letter (for user defined types) follows PascalCase, isn't it?

 0  Reply**Alex** Author Reply to [Ajit Mali](#)¹⁵ 🕒 January 25, 2024 9:42 pm

Yes.

 1  Reply**LinuxCat**

🕒 January 17, 2024 5:50 am

Thanks, very cool.

 0  Reply**Swaminathan**

🕒 January 13, 2024 5:20 am

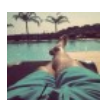
int 3some;

when you want to add fun to work

 20  Reply**zestylegume3000**

🕒 January 11, 2024 12:47 pm

Naa bro felt a lil wild with this one

 12  Reply**Tom**

🕒 January 6, 2024 11:37 pm

great lesson!



0



Reply

**rakshxt**

🕒 December 23, 2023 11:55 pm

wait what

```
1 | int some{};
```

yo author??



19



Reply

**Ground Control**🗨️ Reply to [rakshxt](#)¹⁶ 🕒 January 2, 2024 11:00 am

value is 0 :(



12



Reply

**lofi**🗨️ Reply to [rakshxt](#)¹⁶ 🕒 December 25, 2023 4:53 pm

bro got a little crazy writing that



3



Reply

**Jimmy Himmy**

🕒 December 21, 2023 9:05 am

```
int VALUE; // unconventional (should start with lower case letter)
```

This implies that vALUE might be acceptable.



3



Reply

**Alex** Author🗨️ Reply to [Jimmy Himmy](#)¹⁷ 🕒 December 22, 2023 2:19 pm

True. Amended the reason to be more precise. Thanks!



2



Reply

**oklol**

🕒 December 20, 2023 9:35 am

Good lesson

👍 0

➡ Reply



Ayo1

🕒 December 20, 2023 9:33 am

Good

🔗 *Last edited 3 months ago by Ayo1*

👍 0

➡ Reply



AbeerOrTwo

🕒 December 13, 2023 8:29 pm

int fourSome {};

👍 10

➡ Reply



lol kaimo

🕒 December 3, 2023 1:19 pm

nahhhhhhhh are we not gonna talk about the
int 3some {};

👍 26

➡ Reply



Nerd_Person

🕒 November 14, 2023 5:56 pm

```
int sum {}; // assume it's obvious what we're summing
```

I know it's just a quick quiz, but from my perspective this sounds like the previous developer placed this here assuming people would see it as obvious because it's used as a comment in the question. Maybe use some way that implies that the comment is not apart of the code.

Also love your work! This is helping loads

👍 1

➡ Reply



Thao

🕒 November 4, 2023 7:07 pm

Today is Thank you.



0



Reply

**Krishnakumar**

🕒 October 3, 2023 4:26 am

Thanks for the nice lesson, Alex.

🔗 Last edited 6 months ago by Krishnakumar



0



Reply

**Dominik Wolf**

🕒 September 6, 2023 4:49 pm

```
1 // a count of the number of chars in a piece of text, including whitespace
2 and punctuation
  int numberOfChars;
```

This comment kinda collides with the recommendation you gave us in lection 1.2:

"Third, at the statement level, comments should be used to describe why the code is doing something. A bad statement comment explains what the code is doing. If you ever write code that is so complex that needs a comment to explain what a statement is doing, you probably need to rewrite your statement, not comment it."

But I guess, not everything is as black and white. It kinda depends on the situation so you can do some exceptions.

🔗 Last edited 7 months ago by Dominik Wolf



1



Reply

**Alex**

Author

Reply to [Dominik Wolf](#)¹⁸

September 8, 2023 5:14 pm

The recommendation in 1.2 really pertains more to non-declaration statements. For variables, it's okay to leave a comment indicating what their purpose is. Otherwise, that has to be inferred from the code, which isn't always easy or obvious.



10

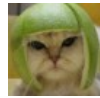


Reply

**noobmaster64**Reply to [Alex](#)¹⁹

September 9, 2023 7:58 am

i like how active the author is

 4  Reply**Alex** Author Reply to [noobmaster64](#)²⁰  September 10, 2023 7:06 pm

Me too.

 45  Reply**Edward** Reply to [Alex](#)²¹  October 2, 2023 8:21 pm

Lol

 1  Reply

Links

1. <https://www.learncpp.com/author/Alex/>
2. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#rules>
3. [javascript:void\(0\)](javascript:void(0))
4. <https://www.learncpp.com/cpp-tutorial/whitespace-and-basic-formatting/>
5. <https://www.learncpp.com/>
6. <https://www.learncpp.com/cpp-tutorial/uninitialized-variables-and-undefined-behavior/>
7. <https://www.learncpp.com/keywords-and-naming-identifiers/>
8. <https://www.learncpp.com/computer-game-programming/introduction-to-roguelike-gaming/>
9. <https://www.learncpp.com/cpp-tutorial/introduction-to-expressions/>
10. <https://gravatar.com/>
11. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-594345>
12. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-593789>
13. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-593494>
14. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-593035>
15. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-592840>
16. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-591312>
17. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-591252>
18. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-586704>
19. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-586802>
20. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-586822>
21. <https://www.learncpp.com/cpp-tutorial/keywords-and-naming-identifiers/#comment-586946>